Journées du Groupe de Travail en Modélisation Géométrique 2018, Aix-en-Provence

Constraints : equations or procedures ?

Dominique Michelucci[1], Jean-Philippe Pernot[2], Marc Daniel[3] et Sebti Foufou[4]

[1]Université de Dijon
[2]ENSAM ParisTech Aix-en-Provence
[3]LSIS Marseille
[4]American University of Abu Dabi

Résumé
Today all CADCAM geometric modellers provide tools for geometric modelling with constraints. Historically, geometric constraints in 3D were represented with equations, and basic problems and systems were solved with some variant of Newton's method, or with the Locus Intersection Method. This article proposes to replace equations with procedures. The modeller calls the solver, which can call every procedures of the modeller. In turn, procedures can call the solver for sub-problems. This approach makes possible things which were impossible, and it eases the specification of CADCAM constraints on composite shapes with heterogenous formats. Computing precise values for derivatives is still possible. Using tools for debugging and decomposing systems into sub-systems or for exploiting sparsity is still possible. Using state-of-the-art numerical solvers and taking advantage of sparsity is still possible. This article presents advantages, inconveniences, arising issues and implementation of this approach.

This article proposes to represent constraints with procedures, instead of equations, each time it is possible. §1 introduces the topic. §2.1 presents existing methods for solving geometric constraints. §2.2 presents existing methods for debugging, decomposing and exploiting sparsity of systems of equations. §2.3 presents existing procedures callable by the solver. §3 presents pros, cons, arising issues of this approach. §3.1 presents advantages, 3.2 presents inconveniences, 3.3 presents issues. §4 presents an implementation. §5 concludes.

1. Introduction

This section presents the big picture. Today, all geometric modellers provide tools for geometric modelling with constraints [HJA05]. §2.1 presents standard methods for solving geometric constraints. §2.2 presents standard methods for the qualitative study of geometric constraints. Most of the time, basic (irreducible) systems of 3D constraints are translated into systems on non linear equations, which are solved with some iterative numerical method, like a variant of Newton's method.

This article proposes to replace equations with procedures, provided by geometric modellers. We use the word "procedure" to avoid the ambiguity of the word "function", both used in Computer Science and in Mathematics. Procedures (§2.3) input known values (*e.g.,* vectors of numeric values), and return values : they compute mathematical functions.

The first and strongest argument for replacing equations with procedures is that procedures are definitively much more convenient and powerful than equations. Geometric modellers on the market provide efficient procedures for generating shapes, and for interrogating shapes and sets of shapes (called assemblies, or scenes). For instance, a typical interrogating procedure computes in an efficient way the point of a given shape S the closest (or the furthest) to a given point p. This procedure may use GPU or clever accelerating data structures : octrees, BSP trees, kd-trees. Suppose now that S is unknown, because its generating parameters are unknown. Suppose also that a point X is in S if $S(X, Y) = 0$ for some system S. To find the point X of S closest to p, we can solve the optimization problem : $\min ||X - p||^2, S(X, Y) = 0$, or we can solve the related Lagrangian system and select the re-

levant root. In practice, if a Newton-like solver is used, it should start close enough from the expected root. However, there is a better method. When a numerical iterative solver is used, it provides (approximate) numerical values for unknown variables (actually modifiable) X and Y at each step. At initialization, X^0, Y^0 are read on the sketch. Thus, after all, it is possible to use procedures of the modeller even when S is unknown (actually, known but modifiable). It would be clumsy to not use these procedures.

A second argument for replacing equations with procedures is that equations are not available in some cases. It is true for shapes like subdivision surfaces and fractals, which have no equation but are results of procedures. This is true when dealing with free-form surfaces which are often obtained tediously from fairly sophisticated modelling functions (e.g. sweep, loft, blend). This is true when specifying constraints on mechanical quantities such as the Von Mises stress which results from a complex Finite Element simulation. Another shape optimization example is the shape of a turbine blade : it is the result of a complex optimization process which aims at finding the best compromise between notably its aerodynamic and mechanical performances.

Representing constraints with procedures instead of equations has many advantages §3.1 and few inconveniences §3.2. §3.3 answers positively to the question : is it still possible to use previous standard methods for debugging, decomposing and solving with this representation of constraints ?

2. Standard methods for geometric constraints

This section summarizes standard methods for solving geometric constraints and for the qualitative study of constraints [HJA05]. Constraints are represented with equations.

2.1. Solving geometric constraints

In classical 2D Euclidean geometric problems solvable with ruler and compass like the emblematic Appolonius's problem, and 3D problems in Monge's descriptive geometry, geometric constraints are incidences, tangencies, distances or angles involving points, lines, circles or conics in 2D, and also planes, spheres or quadrics in 3D (no composite objects !). These problems are straightforwardly translated into systems of polynomial equations [DH00].

Polynomial systems are well known. Many solving methods are available.

Geometric solvers, or constructive solvers, decompose the system of geometric constraints into basic sub-problems, solve them, and assemble solutions. In 2D, the decomposition can be done using rules activated by the inference engine of some expert system [RSV89, VSR92], or with some graph-based bottom-up or top-down method [JTNM06], or with the witness method [MF09]. There is often a formula for solving basic sub-problems (*e.g.,* triangles defined modulo isometry (DMI) by three constraints, quadrilaterals DMI by five constraints, hexagons DMI by six constraints). In 3D, basic sub-problems are much more numerous and difficult, thus other solving methods are needed.

Many basic spatial problems are solvable with the Locus Intersection Method (LIM) : the principle is to remove one constraint, so the solution set becomes a curve, which is followed or sampled until the removed constraint is satisfied [GHY04, FS08, IMS11, ISM14, IMS15]. Geometric solvers, or constructive solvers, and LIM need not only equations, but geometric constraints.

Computer Algebra [Kon92] solves exactly polynomial systems, with Groebner bases, or Wu-Ritt's method, etc. The cost of Computer Algebra is at least exponential, so only small non linear systems are tractable. Computer Algebra is used in pedagogical Dynamical Geometry softwares. It is rarely used in CAD-CAM [Kon92]. Computer Algebra relies on symbolic expressions of polynomial equations. Computer Algebra can simplify or solve non polynomial equations (*i.e.,* using transcendental functions) but not in a guaranteed way.

Interval analysis [Jau01] can isolate each regular root in a given initial box. It is used in Robotics to prove that some mechanisms are correct : they can not get stuck or broken. Interval analysis needs equations to contain the wrapping effect.

Homotopic methods [SWI05, DH00] find all real or complex roots of polynomial systems, and its cost per root is polynomial time. Polynomial equations are needed. Homotopy is popular in Robotics [SWI05]. Homotopy can also be used as a variant of the damped Newton method, which belongs to the next class of numerical iterative methods ; in this case, equations can be replaced with procedures.

Many numerical iterative methods are frequently used to solve systems of non linear equations (possibly non polynomial) : Newton or Newton-Raphson, damped Newton or homotopy, and optimizers : Levenberg-Marquardt (LM) [GCG99a], Broyden-Fletcher-Goldfarb-Shanno's method (BFGS) or limited memory BFGS (lm-BFGS), Hooke-Jeeves (HJ), Nelder-Mead's simplex (NMS) or Torczon's simplex (TS) method. For solving $F(X) = 0$, optimizers typically minimize the norm of the residu $||F(X)||_2^2$. An interest of generalized Newton solvers

(using Least-Square solving)) and of optimizers is that they can solve over-constrained problems. The latter occurs, for example, for computing the intersection point of three circles or three 2D curves. Over-constrainedness is not always a mistake : it can be used to avoid spurious roots. Generalized Newton solvers and optimizers can also solve under-constrained problems.

Numerical iterative methods need a starting point X^0, called the sketch in CADCAM. The sketch is either provided by users (engineers or designers) with some interactive graphical interface, or it is given by a previous release of a product, or it is the result of some algorithm. In CADCAM, at each step of a numerical iterative method, it is possible to visualize the current state of the figure. It usually makes sense for users, who can check that the solver does not go a wrong way, and who can drive the solver. Moreover, many interactive tools enable users to detect and fix errors, like conflicts between constraints. Such debugging tools of qualitative study of constraints are essential.

Strictly geometric constraints are not sufficient for CADCAM. Thus, in late nineties, Hoffmann et al [HJA97], and Joan-Arinyo [JASR99] proposed an hybrid solving method. The idea is to combine a 2D geometric solver and an equational solver. The geometric solver uses the already mentionned constructive approach. It is unsufficient in 3D, but it can be replaced with LIM. Thus, at least in principle, this apprach extends in 3D. This hybrid approach still relies on equations.

BFGS and lm-BFGS solve unconstrained optimization problems. Byrd et al [BLNZ94] propose a variant for solving constrained optimization problems like : $\min G(X)$ with $L \leq F(X) \geq U$.

2.2. Existing tools for qualitative study

It is essential to provide users tools for debugging constraints, detecting errors and fixing them, for decomposing systems into sub-systems, and for exploiting sparsity to speed-up the solving process. This is the qualitative study of constraints. We mention three kinds of tools.

Methods in the first kind call the solver. We call them protocols. Many protocols have been suggested to detect contradicting, or conflicting subsystems. Remember that an over-constrained system can be contradicting or redundant. Here is an example of such a protocol : start from a figure close to the expected solution, and account for (*i.e.,* solve) constraints incrementally. Let E_k be the first non satisfied constraint. Then $C = \{E_1, \dots E_k\}$ is contradicting. Then remove from C every constraint E_i such that $C - E_i$ is still non satisfiable. This protocol gives the smallest contradicting subsystem and can be used when equations are not available. This protocol uses combinatorial properties of dependent / independent sets of constraints, which are formalized by Matroid Theory.

The two other kinds of tools do not call the solver.

The second kind is a set of combinatorial (or structural) methods [Owe91, Ser91, Hen92, JTNM06] which consider various graphs. For simplicity, this article considers only the sparsity graph [Ser91, Hen92]. It is a bipartite graph linking equations (now procedural constraints) and unknowns (now modifiable variables). Combinatorial methods rely on Matching Theory [LP09]. They compute maximum matching in the sparsity graph. They detect structurally under-, over- and well-constrained parts, and structurally under-, over- and well-constrained parts modulo isometries. In both cases, they decompose into structurally irreducible parts. They still can be used when constraints are represented with procedures rather than equations. Consider a procedure computing a function $P : X \in \mathbb{R}^n \to Y = P(X) \in \mathbb{R}^m$. $Y = P(X)$ is structurally or combinatorially equivalent to m equations : $e_i : Y_i - P_i(X) = 0, i = 1, \dots m$. As usual, there is a vertex for each e_i, for each $Y_i, i = 1, \dots m$, and for each $X_k, k = 1, \dots n$. One edge links e_i to Y_i, and n edges link e_i to $X_1, \dots X_n$. Then structural methods apply. They will detect, for example, the structural over-constrainedness of the intersection of three 2D parametric curves (of four parametric surfaces in 3D). But combinatorial methods are limited : they detect only structural conflicts. Moreover a combinatorial characterization of rigidity (well-constrainedness modulo isometries) is still unknown for general geometric constraints and its existence is questionable. The combinatorial characterization of rigidity is the topic of Rigidity Theory. The latter considers only generic point-point distances, which is unsufficient for CADCAM. The next kind of methods overcome these limitations.

The third and last kind of tools is the witness method [MF09] and its variants [TSM$^+$11, MMS14, HKP17]. The principle of these methods is as follows : suppose we want to solve $F(U, X) = U - U_T = 0$, where U are names of parameters (non modifiable variables), U_T is the value of U (T for target), X are names of unknown (modifiable) variables. A witness is a couple U_W, X_W such that U_W and X_W are vectors of numerical values and such that $F(U_W, X_W) = 0$. Moreover it is assumed that the witness is typical of (or even very close [HKP17] to) the target, so that the witness and the target share the same combinatorial properties. More precisely, the ranks of each minor in the known

witness Jacobian $F'(U_W, X_W)$ and in the unknown target Jacobian $F'(U_T, X_T)$ ($F'(U_T, X_T)$ is unknown because exact values of X_T are unknown) are equal. Under mild assumptions of typicality and exactness (consider that for the sake of simplicity, there is no inaccuracy), the witness method detects all dependences between constraints : it is more powerful than structural methods. In practice, users are in charge of deciding the typicality : for instance, a flat triangle (or a flat polyhedron) is not typical of a triangle (of a polyhedron), and this assumption is not an issue. The simplifying exactness assumption is not practicable and is more problematic, but Hao Hu et al [HKP17] recently proposed tools to account for the unavoidable numerical inaccuracy, when using the witness method for modelling free-form curves and surfaces.

All these tools can still be used when constraints are represented with procedures rather than equations. §4.5 explains how the witness method can be used.

2.3. Procedures callable by the solver

In all procedures, variables, also called arguments or parameters, are names of known values. In equations, variables are names of unknowns, which have no value.

The simplest procedures compute simple functions like $\min(a, b)$, $\max(a, b)$, $|a|$. Actually, there is no polynomial system for characterizing $x = \min(a, b)$ for real parameters a and b. In other words there is no polynomial system such that its only root is $x = \min(a, b)$. Indeed, there are always spurious roots. The same holds for $x = \max(a, b)$, for $x = |a|$, for $x = \mathrm{sgn}(a)$ where sgn is the sign of a : $\mathrm{sgn}(0) := 0$ and $\mathrm{sgn}(a) := |a|/a$. Remark that all these functions are equivalent : any one is sufficient to define the others.

A geometrical 1D example where these functions are needed is the signed distance of $a \in \mathbb{R}$ to the segment $[-1, 1]$, which is $|a| - 1$. Another example is when the smallest (or the greatest) root is needed, for solving optimization problem which can not be solved by procedures.

Other simple and convenient procedures compute transcendental functions $\exp, \log, \cos, \sin, \tan, \arctan$, etc. They are needed for helixes or spirals.

The geometric modeller provides many sophisticated procedures, for computing about meshes, surfaces, NURBS, BRep, CSG, etc, for reverse engineering, for performing physical simulations, for generating NC commands, etc.

It is convenient to distinguish generating procedures and interrogating procedures. Generating procedures return shapes : data structures for meshes, NURBS, CSG trees, assemblies, scenes, etc. The degree, the size, the complexity of a shape (for example, the number of vertices, edges, triangles of a mesh, and the topology of the mesh) may vary with values of parameters of the generating procedure. Thus the complexity of a shape may change during solving. Equations do not have such flexibility.

Interrogating procedures compute shape properties, like the smallest or the greatest distance between two given shapes.

3. Pros, cons, issues

3.1. Advantages

It was impossible to constrain sudivision surfaces or fractals with equation, because they have no equation, so specific methods were needed for them ; representing constraints with procedures make that possible, assuming that the procedures of the modeller apply to sudivision surfaces and fractals.

It becomes easy to manage and constrain heterogeneous geometries, like surface subdivision, meshes, trimmed NURBS patches, analytical shapes (quadrics, torii), as shown by the DECO project [GFM$^+$16a], *e.g.,* to impose some G-continuity constraints along contiguous surfaces of different kinds like NURBS and subdivision surfaces.

Using procedures also avoids to uselessly solve optimization problems : for instance, computing a shortest or a greatest distance is made by a procedure of a modeller. Thus it also avoids nested optimizations problems, which are hard to manage.

Another advantage is that it becomes possible to specify constraints on composite figures because modellers's procedures do manage composite figures. A composite figure is composed of several parts, with different topological dimensions : points, segments or pieces of curves, surfaces patches, volumes pieces. A segment is the simplest composite object. A mesh or a BRep are composite objects. In passing, classical geometric problems (like Appolonius' problem) never consider composite objects, because there is no more geometric constructions with ruler and compass. In CADCAM, composite objects are essential : CADCAM shapes, assemblies and scenes are always composite objects.

We explain now why it is impossible, or difficult, to specify constraints on composite figures with equations. If it is straightforward to pose a system of equations equivalent to : $A(X) = 0$ and $B(X) = 0$ (it is just $A(X) = B(X) = 0$), it is more problematic to pose a system of equations equivalent to : $A(X) = 0$

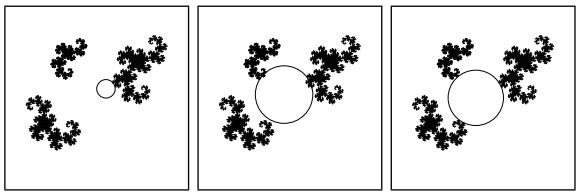

Figure 1: *Generalization of the Appolonius problem to three arbitrary objects. The figure shows the first three iterations of BFGS. They suffice to visually converge to a solution.*

or $B(X) = 0$, which is needed for computing the distance of a given point to a composite figure, or equivalently, for computing which point of which part of the composite figure is the closest to the given point. One may think to $||A(X)|| \times ||B(X)|| = 0$, but the risk is high for the iterative solver to be trapped by a spurious root (a local maximum) or a local minimum. In comparison, a procedure finds the exact result, and is faster.

Resorting to procedures avoids inconveniences of systems of non-linear equations : there is an exponential number of roots but only real roots close to the sketch are relevant. There is no polynomial system characterizing $x = \min(a, b)$ where a and b are real parameters. Idem for $x = \max(a, b)$, for $x = |a|$, because there are always spurious roots. Worse, for some geometric problems, there are continuums of degenerate, spurious roots (*e.g.,* flat solutions instead of 3D solutions), and deflation methods are needed.

The implementation of the proposed approach does not require the development of a new type of modellers, contrarily to the DECO project [GFM$^+$16b].

Another advantage is that the solver can use a large set of very sophisticated and efficient geometric procedures available in the modeler or in the PDP software. Few examples of such procedures are the computations of distances, furthest distances, interpenetration depths, bounding boxes, volumes, intersections, Boolean operations between solids, blending, filleting, meshing, reconstructing, etc. In the classical approach, constrained distance can only be distances between simple elements (points, lines or planes), whereas with procedural constraints, it can be distances between complex shapes like assemblies. Figure 1 shows the generalization to three arbitrary objects of Appolonius problem which consists in finding the circle tangent to three given circles. Procedural constraints permit to generalize to three arbitrary objects, solving with BFGS : $d_A(x, y) - R = d_B(x, y) - R = d_C(x, y) - R = 0$. The procedure $d_A(x, y)$ computes the distance from point (x, y) to the object A, whereas x, y and R are the unknowns. It is possible to generalize using greatest distances D_A, D_B, D_C and solving : $(d_A(x, y) - R)(D_A(x, y) - R) = (d_B(x, y) - R)(D_B(x, y) - R) = (d_C(x, y) - R)(D_C(x, y) - R) = 0$.

Integrating a procedural constraints solver into an existing parametric modeller, such as FreeCAD [Frea] and FreeSHIP [Freb] for CAD, or Blender [Ble] for computer graphics, brings several low-cost advantages such as : (i) improving the functionalities of the modeller, (ii) opening more possibilities for the modeller in terms of constraint formulation and solving, (iii) simplifying the solver as well as the modeller in terms of functionalities and usage.

Next section considers inconveniences of resorting to procedures.

3.2. Inconveniences

This section discusses inconveniences of representing constraints with procedures, instead of equations.

A true inconvenience is that Computer Algebra can no more be used. For example Buchberger's method considers equations as rewriting rules : $x^2 - 2 = 0$ is converted into the rule : $x^2 \to 2$. It is no more possible when equations are not available (assuming they are polynomial).

Similarly, interval analysis [Jau01] can no more be used. Interval arithmetic can still be used. However, it is hard to contain the wrapping effect when equations are not available. Interval arithmetic can be used only for initially sharp intervals, *e.g.,* for computing tolerances.

To prevent a possible misunderstanding, remark that procedures of the modeller can still use and call subdivision solvers [EK01, FM12], typically for solving geometric sub-problems involving free-form curves, surfaces and volumes. Basically, procedures only need to know numerical values of coordinates of control points, in order to call subdivision solvers. Subdivision solvers rely on well-known geometric properties of Bernstein bases and of splines bases, like the convex hull property and the variation diminishing property. For the same reasons, procedures can also use interval analysis for solving geometric subproblems.

Another inconvenience is due to non analytic functions (splines, NURBS, $\min, \max, |.|$, functions using if-then-else instructions). These functions are non polynomial. They are essential in CADCAM : it is not an option to not account for non analytic functions. Procedure compute them in a straightforward way, contrarily to equations. But using non analytic functions has

two consequences. Only the second is indeed an inconvenience.

First consequence, deciding dependences of non analytic functions on variables (does $F_l(X)$ depends on X_c ?) is non decidable. It is decidable in a probabilistic way for analytical functions, but slow. Thus the best solution is that the modeller informs the solver of the dependences, in other words which entries in the Jacobian are not structurally zero. Call this information the sparsity data or the sparsity graph. It is equivalent to the classical bipartite graph linking equations (now procedures) and unknowns (now modifiable variables). This graph is used to detect the structurally over-, under- and well-constrained part, and the irreducible well-constrained subsystems in the well-constrained part. Methods in sparse linear algebra also rely on this graph [Kho12]. Today, the interface of some non linear solvers or optimizers does not account for the sparsity data. Assuming that the sparsity data is available, it is still possible (4.5)to exploit the sparsity of systems of constraints in CADCAM, *i.e.,* the fact that the results of procedures do not depend on all unknown (actually modifiable) variables.

Second consequence, homotopy solvers which find all roots of polynomial systems are no more usable, for two reasons : first, equations are no more available, and second, even if they were (with DAG), functions computed by procedures are non analytical, thus non polynomial, so homotopy theory no more applies. For example, the fundamental theorem of algebra (a degree d polynomial has d complex roots, *i.e.,* $\mathcal{C}$ is an algebraically closed field) does not apply to piecewise polynomials.

3.3. Issue : using pre-existing methods ?

The main issue is : is it possible to use standard, classical methods when equations are replaced with procedures ? §3.3.1 considers the computation of derivatives. §3.3.2 considers this issue for the qualitative study. §4 presents an implementation, which shows that solving with procedures is as fast as solving equations.

3.3.1. The derivatives computation issue

An apparent inconvenience of prefering procedures to equations seems that, since the expression of the left hand side (LHS) of equations is no more available, it is impossible to compute the symbolic expression of derivatives (*i.e.,* if $f(x) = x^2$, then $f'(x) = 2x$). It is still possible to numerically compute finite differences, but it may be too inaccurate, especially for the witness methods (which computes ranks of Jacobian minors). It is still possible to use symbolic differentiation, but the latter has limitations, for if-then-else instructions, for loops, for recursion, for $\min, \max, |.|$, etc, and is not easy to use. A significant result is that automatic differentiation, with the arithmetic of dual numbers [MF09, Fis17], computes precise (with the accuracy of floating-point arithmetic) values of derivatives at a given point ($f'(3) = 6$ if $f(x) = x^2$). For instance, if a shape depends on, say, a small number $n = 6$ of parameters $U = (u_1, \dots u_n)$, a FEM simulation using the dual numbers arithmetic and computing a performance $p(u_1, u_2, \dots u_n)$ will automatically compute in the same time (assuming n is a small constant) $p(U)$ and all derivatives, *i.e.,* the gradient $\nabla p = (\partial p/\partial u_i(U))$, making possible to search the optimal value U^* which maximizes the performance. Even if the procedure computing the performance $p(U)$ generates for some physical FEM simulation some temporary and huge mesh with $N \gg n$ 3D vertices, *e.g.,* $N = 10^5$, or any other huge data structure, only n dual numbers or infinitesimals are needed, and not $n + 3N$.

Thus, after all, it is still possible to compute precise values (but not symbolic expressions) of derivatives and gradient vectors when constraints are represented with procedures.

3.3.2. The qualitative study issue

An issue is : is it still possible to use existing tools for the qualitative study of systems of constraints ? The answer is positive : §4.4 explains how to build the sparsity graph. §4.5 explains how to use the witness method.

4. Implementation

4.1. The new architecture

The modeller calls the solver. The solver can call every procedures, either common and simple (but anavoidable) procedures like $\min, \max, |.|, \cos, \arctan$, etc, and also all more sophisticated procedures provided by the modeller (or by modellers) : procedures for creating and for interrogating shapes, or non geometric objects. Symetrically, procedures of the modeller can call the solver for sub-problems.

Of course, procedures should be correct and consistent. For instance, arguments of a generating procedure should be independent.

A technical issue is the need for some book-keeping. Procedures often return points or vectors in $\mathbb{R}^n$. For instance the procedure computing a point on a Bézier curve $B(P, d, t)$ (P is the vector of control points, d the degree, t the parameter) returns the 3D point, and its Frénet frame. It is convenient to define procedures to access each of the fields, each of the coordinates

(getX, getY, getZ [GFM+16a]). But it is clumsy to call several times the procedure B for the same values of its arguments. Some book-keeping takes care of that [GFM+16a]. Either some memorization (called memoization in functional programming) is used, or the solver calls some handle function to enable the modeller to update the geometric model, before the solver evaluates procedural constraints with calls to interrogating procedures of the modeller.

A technical issue [GFM+16a] is that some arguments of procedures have bounds, for example $t \in [0, 1]$ for a Bézier curve $C(t)$. Many solutions are possible when t is outside its range. The procedure which computes the point $C(t)$ can compute it when t is outside $[0, 1]$: indeed it does exist and is well defined. The procedure can also clamp the value of t inside $[0, 1]$ (but without modifying the variable t of the solver). Another solution is that it is the solver which accounts for bounds on variables. Thus the modeller must provide these bounds to the solver. Either the solver clamps t only, or it clamps all the vector ΔX so that the variable t stays inside $[0, 1]$. In combination to all these methods, it is possible to also use the procedural constraint : $(t - 1/2)^2 + t_s^2 - 1/4 = 0$ where t_s is a new, slack variable of t. We did not try penalties. Byrd et al [BLNZ94] proposed a more sophisticated method to handle bounds constraints, actually to solve constrained optimization with lm-BFGS.

4.2. Interface modeller-solver interface

The interface, or the communication protocol, between the modeller and the solver must permit to benefit from the sparsity of procedural constraints. There are mainly two kinds of interfaces. The first interface is used in the DECO project [GFM+16a] : the modeller and the solver use black DAG (Directed Acyclic Graphs). This first interface permits to exploit the sparsity of procedural constraints. However, many libraries such as GNU GSL or Scipy, which provide solvers or minimizers, use the second kind of interface, as follows. To solve a system $F(U, X) = 0$ with $U = U_T$, the solver typically receives three arrays F, U, X : F is an array of pointers on procedures computing $F(U, X)$ (equations are : $F(U, X) = 0$), U is the array of floating-point values for parameters, X is the array of initial values for the unknowns X. The solver can modify X, but not U. Sometimes the solver accepts an array (of pointers on procedures) F' for computing the gradient vectors ∇F_i. It can also use finite differences to approximate derivatives. The solver also receives technical information such as array sizes, threshold values for termination tests, a maximum number of iterations, etc. The same kind of interface applies for constrained or unconstrained minimization problems. Finally, the solver receives some handle methods, *e.g.,* for drawing pictures of the current figure in our context ; it can permit users to monitor and drive the resolution process. Clearly, this second interface is not sufficient for exploiting sparsity. Another array D is needed for specifying dependences, *i.e.,* the sparsity graph : $D[c]$ is the list of all (index of) variables X_k on which the constraint F_c depends. D can be seen as a sparse matrice, and its transpose T can be used instead : $T[k]$ is the list of (index of) constraints depending on X_k. These two arrays are clearly equivalent. They are sufficient to exploit sparsity. They are already used for that purpose in the interface of many Sparse Linear Algebra libraries [Saa03], for instance to compute fill-reducing ordering of unknowns. These arrays represent the sparsity graph, *i.e.,* the bipartite graph equations-unknowns used in matching theory (Dulmage-Mendelsohn decomposition).

4.3. Mathematical conditions

As usual, functions computed by procedures must be smooth (like a distance function) or smooth almost everywhere (like the closest point function, *i.e.,* the orthogonal projection of a given point on a given shape).

This mathematical condition can sometimes be relaxed. For instance Gouaty et al [GFM+16a] computes constrained gearwheels : the number of teeth must be an integer. Similarly for the number of steps of a staircase. Actually, gearwheels, staircases, steel structures, wooden carpentry, etc are algorithmic shapes : their generating algorithms are standardized in technical documents and can be straightforwardly implemented in procedures.

As usual, the sketch should be close enough to the expected root.

The modeller must provide the sparsity data to the solver, for exploiting sparsity of the system, and for the qualitative study of the system.

It is possible to still use the witness method (4.5). As usual, the modeller must provide the solver some metadata : maximal number of iterations, tolerances for termination test or computations of ranks, a callback procedure which displays the current state of the figure, which algorithm to use for solving, etc.

4.4. Building the sparsity graph

We explain here how the sparsity graph, equivalent to the standard bipartite graph equation-unknown, can be constructed. Consider a procedure computing a function $P : X \in \mathbb{R}^n \to Y = P(X) \in \mathbb{R}^m$. Combinatorially, it is equivalent to m equations : $Y_i - P_i(X) = 0, i = 1, \dots m$.

4.5. Interface for using the witness method

For the witness method to apply, the modeller (or any procedure calling the solver) must provide a witness : either a previous release of the product, or a computed witness [KMF14]. It is the users who decide if the witness is typical of the expected solution. Also, for enabling the witness method to detect rigid subsystems, variables must be tagged so that the witness method can compute a priori a base for infinitesimal (again, dual numbers appear [Fis17]) rigid body motions ([Fis17] for dual numbers, [MF09] for the witness method). Tags give the kind of coordinates of the tagged variable when it is a coordinate ; remember that values of coordinates depend on the used Cartesian frame. A tag, *i.e.,* a coordinate can be :

- the x, the y or the z of a point. The x, y, z of the same point must be linked together in some way, *e.g.,* the variable X_3 is the x of the point (X_3, X_4, X_5).

- the x, the y or the z of a vector. Indeed, vectors and points are different because they do not behave the same under translations. The x, y, z of the same vector must be linked in some way.

- the a or b or c of a normal vector. Vectors and normal vectors are different : a vector is the difference between two points. A normal vector is associated to a linear form : $(x, y, z)^t \to (a, b, c)(x, y, z)^t$. Actually, the a, b, c is the vectorial part of the equation of a plane (see below). The a, b, c of the same normal vector must be linked in some way.

- the a or b or c or d of the equation of a plane : $ax + by + cz + d = 0$. The a, b, c, d of the same plane must be linked together.

- a geometric variable which is independent on the Cartesian frame : radius or length, area, volume, scalar product.

- finally a variable can be a non geometric variable, thus independent on the Cartesian frame : energy, force, cost, etc.

The last two tags can be merged. Tags permit the witness method to decompose systems into rigid (*i.e.,* well-constrained modulo isometry) sub-systems. It is also possible to tag variables to account for similitudes but this decomposition is more rare [SS06, SM06].

4.6. Dual numbers

Derivatives are key components in many geometric computations. This section introduces dual numbers and shows how they can be used to compute derivatives exactly, with the accuracy of floating-point arithmetic, even when equations are not available. This accuracy is needed by the witness method.

Dual numbers are best understood with an analogy with complex numbers ($\mathcal{C}$). For a computer scientist, writing a C++ library for an arithmetic for dual numbers and for complex numbers is almost the same. A complex number z is a pair of two real numbers ($x \in \mathbb{R}, y \in \mathbb{R}$). The pair $(0, 1)$ is called i. The part x of z is its real part, and y its imaginary part. The addition of two complex numbers $z = (x, y)$ and $z' = (x', y')$ is defined by

$$(x, y) + (x', y') = (x + x', y + y')$$

Thus it is consistent to write the pair $z = (x, y)$ as $(x, 0)+(0, y) = x(1, 0)+y(0, 1) = x1+yi$. The product of z and z' is defined by

$$(x, y) \times (x', y') = (xx' - yy', xy' + yx')$$

thus $i^2 = (0, 1) \times (0, 1) = (-1, 0) = -1$. Actually, reducing i^2 to -1 gives another path to the product rule :

$$\begin{aligned}(x + yi) \times (x' + y'i) &= xx' + yy'i^2 + (xy' + yx')i \\ &= (xx' - yy') + (xy' + yx')i\end{aligned}$$

There is a remarkable isomorphism $\phi_{\mathcal{C}}$ between $z \in \mathcal{C}$ and the 2×2 real matrice

$$\phi_{\mathcal{C}}(z) = \begin{pmatrix} x & -y \\ y & x \end{pmatrix}$$

Indeed, $\phi_{\mathcal{C}}(z + z') = \phi_{\mathcal{C}}(z) + \phi_{\mathcal{C}}(z')$ and $\phi_{\mathcal{C}}(z \times z') = \phi_{\mathcal{C}}(z) \times \phi_{\mathcal{C}}(z')$.

A dual number ressembles a complex number. It is a pair of two real numbers ($x \in \mathbb{R}, y \in \mathbb{R}$). The pair $(0, 1)$ is called ϵ and can be thought as an infinitesimal number. x is the standart part of the pair (x, y), and y its infinitesimal or non standard part. The addition of two dual numbers (x, y) and (x', y') is defined by

$$(x, y) + (x', y') = (x + x', y + y')$$

Thus it is consistent to write the pair $z = (x, y)$ as $(x, 0)+(0, y) = x(1, 0)+y(0, 1) = x1+y\epsilon$. The product of $z = x + y\epsilon$ and $z' = x' + y'\epsilon$ is defined by

$$(x, y) \times (x', y') = (xx', xy' + yx')$$

thus $\epsilon^2 = (0, 1) \times (0, 1) = (0, 0)$. Actually, reducing ϵ^2 to 0 gives another path to the product rule :

$$\begin{aligned}(x + y\epsilon) \times (x' + y'\epsilon) &= xx' + (xy' + yx')\epsilon + yy'\epsilon^2 \\ &= xx' + (xy' + yx')\epsilon\end{aligned}$$

This time, the isomorphism ϕ between the dual number $z = x + y\epsilon$ and the 2×2 real matrice $\phi(x + y\epsilon)$ is defined by :

$$\phi(z) = \begin{pmatrix} x & 0 \\ y & x \end{pmatrix}$$

Indeed, $\phi(z + z') = \phi(z) + \phi(z')$, $\phi(z \times z') = \phi(z) \times$

$\phi(z')$, thus $\phi(1/z) = \phi(z)^{-1}$ and $\phi(z^k) = \phi(z)^k$. Let us detail the product :

$$\begin{array}{ccccc} (a+b\,\epsilon) & \times & (a'+b'\,\epsilon) & = & aa' + (ab'+ba')\,\epsilon \\ \downarrow & & \downarrow & & \\ \begin{pmatrix} a & 0 \\ b & a \end{pmatrix} & \times & \begin{pmatrix} a' & 0 \\ b' & a' \end{pmatrix} & = & \begin{pmatrix} aa' & 0 \\ ba'+ab' & aa' \end{pmatrix} \end{array}$$

From this isomorphism, we deduce that :

$$\frac{1}{a+b\,\epsilon} = \frac{1}{a} - \frac{b}{a^2}\,\epsilon \text{ when } a \neq 0$$

thus $b\epsilon$ has no inverse (the associated matrix is not invertible). This rule is a special case of :

$$(a+b\epsilon)^k = a^k + ka^{k-1}b\,\epsilon$$

If P is a polynomial, then $P(x_v + \epsilon)$ where x_v is a floating-point number, gives $P(x_v)$ and the derivative $P'(x_v)$:

$$\begin{aligned} P(x_v + \epsilon) &= a(x_v+\epsilon)^3 + b(x_v+\epsilon)^2 + c(x_v+\epsilon) + d \\ &= a(x_v^3 + 3x_v^2\,\epsilon) + b(x_v^2 + 2x_v\,\epsilon) + c(x_v + \epsilon) + d \\ &= (ax_v^3 + bx_v^2 + cx_v + d) + (3ax_v^2 + 2bx_v + c)\,\epsilon \\ &= P(x_v) + P'(x_v)\,\epsilon \end{aligned}$$

It extends to multivariate polynomials : either we have only one ϵ and two evaluations are needed :

$$\begin{aligned} Q(x_v+\epsilon, y_v) &= Q(x_v, y_v) + Q'_x(x_v, y_v)\epsilon \\ Q(x_v, y_v+\epsilon) &= Q(x_v, y_v) + Q'_y(x_v, y_v)\epsilon \end{aligned}$$

or each variable is attached its own ϵ and one evaluation suffices :

$$\begin{aligned} Q(x_v+\epsilon_x, y_v+\epsilon_y) \;=\; & Q(x_v, y_v) + Q'_x(x_v, y_v)\epsilon_x + \\ & Q'_y(x_v, y_v)\epsilon_y \end{aligned}$$

Actually, this feature (computing $(P(x), P'(x))$ together) extends to non polynomial functions. The arithmetic of dual numbers is used to compute $f(t)$ and its derivative $f'(t)$ in the same time : here f is a continuous and derivable function (at least at value t), and $f'(t)$ is the value of the derivative of f at $t \in \mathbb{R}$. The idea is to represent the pair $(f(t), f'(t))$ with a dual number number $f(t) + f'(t)\epsilon$. It is possible because the arithmetic of dual numbers mimics the rules for derivatives of sums and products (caution : the primes in f', g' denote the derivatives) :

$$(f, f') + (g, g') = (f+g, f'+g')$$

and

$$(f, f') \times (g, g') = (f \times g, f \times g' + f' \times g)$$

In a library for complex numbers, we have to define $\exp, \cos$, etc for complex numbers. Idem for dual numbers. This definition is straightforward : for any f, $f(a+b\epsilon)$ equals by definition $f(a) + b\epsilon f'(a)$, thus :

$$\begin{aligned} \exp(a+b\epsilon) &= e^a + be^a\,\epsilon \\ \cos(a+b\epsilon) &= \cos(a) - b\sin(a)\,\epsilon \\ \sin(a+b\epsilon) &= \sin(a) + b\cos(a)\,\epsilon \\ \tan(a+b\epsilon) &= \tan(a) + b(1+\tan^2(a))\,\epsilon \end{aligned}$$

The definition of the discontinuous function $\mathrm{sgn}(a + b\epsilon)$ is easy :

$$\mathrm{sign}(a+b\epsilon) = \mathrm{sign}(a) + (1 - \mathrm{sign}(a)^2)\mathrm{sign}(b)$$

where $\mathrm{sgn}(v \in \mathbb{R})$ is -1 if $v < 0$, 1 if $v > 0$ and 0 if v is zero. Then the definition of the absolute value follows :

$$|a+b\epsilon| = \mathrm{sign}(a+b\epsilon) \times (a+b\epsilon)$$

The definitions of the *min* and *max* of two dual numbers could be formulated in a similar way. In passing, symbolic differentiation (at compile time) can not deal so nicely with functions $|.|, \min, \max$ and if-then-else constructs.

Dual numbers permit to compute the derivative of $D(X) = \det(M(X))$, for square matrices $M(X)$, even if entries of M are piecewise polynomials, or algorithms : just replace floating point numbers with dual numbers and then use any standard numerical method (Gauss pivot, LUP). There are also formulas.

Lemma : $\det(I + \epsilon M) = 1 + \mathrm{Trace}(M)\,\epsilon$, where M is standard :

$$\begin{aligned} \det(I+\epsilon M) &= (1+M_{11}\epsilon)(1+M_{22}\epsilon)\ldots(1+M_{nn}\epsilon) + R \\ &= 1 + \mathrm{Trace}(M)\,\epsilon + R \end{aligned}$$

where R represents other perfect matching in $I + \epsilon M$. But other perfect matching use at least two off-diagonal entries in $I + \epsilon M$, thus are multiples of ϵ^2, thus are zero.

When A is invertible, $\det(M(x+\epsilon)) = \det(A + \epsilon B)$ is :

$$\begin{aligned} \det(A+\epsilon B) &= \det(A(I + \epsilon A^{-1}B)) \\ &= \det(A)\det(I + \epsilon A^{-1}B) \\ &= \det(A)(1 + \mathrm{Trace}(A^{-1}B)\,\epsilon) \end{aligned}$$

When A is not invertible, we use its SVD : $A = U\Sigma V^t$ (with Σ diagonal and U, V unitary) :

$$\begin{aligned} \det(A+\epsilon B) &= \det(U\Sigma V^t + \epsilon B) \\ &= \det(U(\Sigma V^t + \epsilon U^t B)) \\ &= \det(U(\Sigma + \epsilon U^t B V)V^t) \\ &= \det(\Sigma + \epsilon U^t B V) \end{aligned}$$

equals the product of diagonal entries of $\Sigma + \epsilon U^t B V$. It is 0 when there are at least two null singular values in Σ. Otherwise it is

$$(\sigma_1 + k_1\epsilon)\ldots(\sigma_{n-1} + k_{n-1}\epsilon)(0 + k_n\epsilon) = 0 + \sigma_1\ldots\sigma_{n-1}k_n\,\epsilon$$

The main mathematical difference between complex numbers and dual numbers is that complex numbers form a (commutative) field, while dual numbers only form a commutative ring : they are not a field because there are non zero divisors of zero (ϵ and all $b\epsilon$ for $b \in \mathbb{R}$).

We conclude this section by showing the relevance of dual numbers for geometric computations. An algebraic construction ϕ starting from $\mathbb{R}$ gives the quaternions, which represents 3D rotations. If ϕ is applied to $\mathbb{R} + \epsilon\mathbb{R}$, it gives biquaternions, aka dual quaternions, which represents both 3D rotations and translations.

4.7. Solving without equations

This section shows the feasibility and the scalability of solving without equations.

The solver must solve without equations the underlying systems $E(X) = 0$, starting from an initial guess close enough to the expected root. This proximity was already needed with classical geometric constraints, *i.e.,* when equations are available. In practice, users provide this initial guess with a sketch.

We implemented and tested several solving methods to be used in the proposed approach : Newton, Levenberg-Marquardt [GCG99b], lm-BFGS (Limited memory Broyden-Fletcher-Goldfarb-Shanno) [BGLS06], Hooke-Jeeves, stochastic descent and the Jaya heuristic [Rao16]. Some methods need gradients. They are computed with centered finite difference (not with dual numbers). Methods have been tested on this geometric problem : let ABC be a given triangle ; let r be an integer ; place r rows of circles in ABC, with one circle in the first row, k circles in row k, so that circles are outwards tangent to their neighbors or to the sides of ABC. See Figures 2 and 3. For r rows, there are $r(r+1)/2$ circles and 3 times more unknowns : $n = 3r(r+1)/2$ as each circle k has unknown center coordinates (x_k, y_k) and unknown radius r_k. This is a well-constrained problem as the number of constraints is equal to the number of unknowns $n = 3r(r+1)/2$. It is also irreducible : it has no well-constrained (nor rigid) subsystem. Though equations (circle-circle tangencies, or line-circle tangencies) are available, they are not accessible to the solver : the solver is only aware of an array of n (pointers on) procedures $E[0, \dots n-1]$, and an array of n unknowns V which are the concatenation of tuples (x_k, y_k, r_k) for $k = 0, \dots n-1$. The initial values for (x_k, y_k, r_k) are obtained as follows : this problem is easy to solve when the triangle ABC is equilateral, which gives barycentric coordinates a_k, b_k, c_k (with $a_k + b_k + c_k = 1$) for the center of circle k : $(x_k, y_k) = a_k A + b_k B + c_k C$. They give an initial guess for X.

This problem is artificial but very convenient for measuring performances, and controlling the behaviour and output of algorithms, due to its visual and intuitive nature. Figure 2 shows Newton iterations for 11 rows, and Figure 3 some Hooke-Jeeves iterations. Table 1 shows the figures with more rows, and table 2 gathers together the empirical complexities of the tested algorithms. The empirical complexity for an algorithm is the slope of the curve (very close to a line) of the log-log diagram $(\log n_i, \log T(n_i))$, where n_i is the number of unknowns, $T(n_i)$ is the running time of the algorithm. We tested with 50, 100, 150, 200 rows. lm-BFGS has the best empirical complexity : $O(n^{1.33})$. Then Newton and Levenberg-Marquardt have complexity $O(n^{1.4})$. Hooke-Jeeves has complexity $O(n^{1.9})$. All these complexities are less than $O(n^2)$, the size of a dense Jacobian. To reach them, it is essential to exploit the sparsity of systems of procedural constraints. Remember that, for multiplying two dense matrices, or solving a dense linear system, the complexity of the famous Strassen method is $O(n^{2.807})$, and the complexity of the Coppersmith-Winograd algorithm (the best method known so far, but unused in practice) is $O(n^{2.376})$. Even the Hooke-Jeeves method is much better, and lower than $O(n^2)$. Other methods : stochastic gradient, and Jaya either diverge, or converge with damping but are too slow, so we will not comment their complexity.

4.8. About meta-heuristics

In our experiments, classical and heuristics enhanced solvers such as Jaya and stochastic descent did not perform well with the test problem. However, turning to meta-heuristics could be interesting when there is a huge set of solutions for which users have no a priori preference, but they can reject a bad solution, even if it is difficult for them to say why ; or it is too time consuming to explicit all constraints and preferences. A similar problem is met in forensic : some witnesses can not describe faces or persons, though they can recognize them. To solve this forensic issue [SGM09], a software (like EvoFIT) generates a set of 20 pictures of random faces ; the witness chooses the faces which resemble the most the target face. The software combines parts of selected faces, proposing a new set of 20 faces to the witness. After some rounds, a resemblant portrait is reached. One may imagine to use the same method for choosing a solution amongst a huge set. The modeller generates 20 solutions, for instance starting from 20 random seeds, improved with some Newton iterations or some minimization ; users reject too bad solutions ; maybe they can say which part is good in some solution and should be kept. The modeller combines selected solutions, as in genetic algorithms, and improves each new combination to satisfy

r	50	100	150	200	300	400
c	1275	5050	11325	20100	45150	80200
u	3825	15150	33975	60300	135450	240600

Table 1: *Number of rows r, circles c and unknowns u for the test problem.*

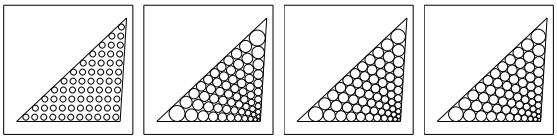

Figure 2: *Solving with Newton (11 rows). Three iterations suffice.*

the procedural constraints, with Newton iterations or a Minimizer. One may expect to find a good solution after some rounds.

5. Conclusion

This article proposes to represent constraints with procedures, and it presents the advantages, the few inconveniences, and an implementation which proves the feasibility and the scalability of this approach.

For conciseness, we cannot detail some precursing works which inspire us [GFM+16b], [DPD15], [PFGL08a], [GMMP11], [FGLP14], [PFGL08b], [LGP+16].

We conclude with future works.

A more precise formalization is needed, to explicit links with PLM, ontologies, shape analysis, etc.

A straightforward extension of our work is to manage an hybrid representation for geometric models : basically a tuple (X^T, U^T, F) or in words, a geometry and a set of constraints. The geometry X^T (T like target) is a solution of $F(U, X) = 0, U - U^T = 0$, F is the set of constraints, represented with DAG or the source of an equivalent program in some script embedded language (Lua, Python, Lisp), U is the vector of parameters, U_T is the vector of parameters values, X is the vector of unknown variables. X^T, U^T is its own witness. There are many ways to modify and update the model.

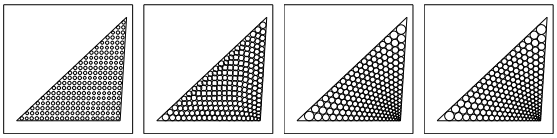

Figure 3: *Solving with Hooke-Jeeves (20 rows).*

Algorithm	Complexity
L-BFGS ($m = 10$)	$O(n^{1.33})$
Newton	$O(n^{1.4})$
Levenberg-Marquardt	$O(n^{1.4})$
Hooke-Jeeves	$O(n^{1.9})$

Table 2: *n is the number of circles or of unknowns and constraints (and not the number of rows). This table gives the empirical complexity (the slope of log-log diagram) of algorithms for the test problem, exploiting sparsity.*

The first way is to modify values of $U : U = U^*$ instead of $U = U^T$, and X must be updated with some solving. A possibility is to follow an homotopy curve with equation : $H(t, X) = (1 - t)F(U^T, X) + tF(U^*, X) = 0$ from $t = 0, X = X^T$ to (hopefully) $t = 1$, $X = X^*$, or to minimize $||H(t, X)||$ from $t = 0, X = X^T$ to (hopefully) $t = 1$, $X = X^*$.

The second way is to specify new geometric elements and new constraints : this part is well known, since it is standard to interactively and incrementally add geometric elements and constraints to the sketch, to detect and fix errors, and to solve.

The third and last way to modify the model is to interactively modify the geometry X^T. Sometimes it is much easier for users to interactively move vertices or faces (or tangent planes) than to change constraints. Updating U values does not suffice in some cases, and constraints F must be updated or reverse-engineered : it is clearly the hard part.

References

[BGLS06] Joseph-Frédéric Bonnans, Jean Charles Gilbert, Claude Lemaréchal, and Claudia A Sagastizábal. *Numerical optimization : theoretical and practical aspects.* Springer Science & Business Media, 2006.

[Ble] Blender.org. Blender for Open source 3D Creation.

[BLNZ94] Richard H. Byrd, Peihuang Lu, Jorge Nocedal, and Ciyou Zhu. A limited memory algorithm for bound constrained optimization. *SIAM JOURNAL ON SCIENTIFIC COMPUTING*, 16 :1190–1208, 1994.

[DH00] Cassiano Durand and Christoph M. Hoff-

mann. A systematic framework for solving geometric constraints analytically. *Journal of Symbolic Computation*, 30(5) :493 – 519, 2000.

[DPD15] Dorian Decriteau, Jean-Philippe Pernot, and Marc Daniel. Towards declarative CAD modeler built on top of a CAD modeler. *Proceedings of CAD'15, Computer Aided Design and Applications*, pages 107–112. , jun 2015.

[EK01] Gershon Elber and Myung-Soo Kim. Geometric constraint solver using multivariate rational spline functions. In *Proceedings of the sixth ACM symposium on Solid modeling and applications*, pages 1–10, New York, NY, USA, 2001. ACM.

[FGLP14] Bianca Falcidieno, Franca Giannini, Jean-Claude Léon, and Jean-Philippe Pernot. Processing free form objects within a product development process framework. In Jonh G. J. G. Michopoulos, Christiaan J.J. Paredis, David W. Rosen, and Judy M. Vance, editors, *Advances in Computers and Information in Engineering Research*, volume 1, pages 317–344. ASME-Press, 2014.

[Fis17] Ian Fischer. *Dual-number methods in kinematics, statics and dynamics*. Routledge, 2017.

[FM12] Sebti Foufou and Dominique Michelucci. Bernstein basis and its application in solving geometric constraint systems. *Journal of Reliable Computing*, 17 :192–208, 2012.

[Frea] FreeCAD. FreeCAD : An open-source parametric 3D CAD modeler.

[Freb] FreeSHIP. FreeSHIP : Surface Modeling.

[FS08] Arnaud Fabre and Pascal Schreck. Combining symbolic and numerical solvers to simplify indecomposable systems solving. In *Proceedings of the 2008 ACM Symposium on Applied Computing*, SAC '08, pages 1838–1842, New York, NY, USA, 2008. ACM.

[GCG99a] Jian-Xin Ge, Shang-Ching Chou, and Xiao-Shan Gao. Geometric constraint satisfaction using optimization methods. *Computer-Aided Design*, 31(14) :867–879, 1999.

[GCG99b] Jian-Xin Ge, Shang-Ching Chou, and Xiao-Shan Gao. Geometric constraint satisfaction using optimization methods. *Computer-Aided Design*, 31(14) :867 – 879, 1999.

[GFM+16a] Gilles Gouaty, Lincong Fang, Dominique Michelucci, Marc Daniel, Jean-Philippe Pernot, Romain Raffin, Sandrine Lanquetin, and Marc Neveu. Variational geometric modeling with black box constraints and DAGs. *Computer-Aided Design*, 75 :1–12, 2016.

[GFM+16b] Gilles Gouaty, Lincong Fang, Dominique Michelucci, Marc Daniel, Jean-Philippe Pernot, Romain Raffin, Sandrine Lanquetin, and Marc Neveu. Variational geometric modeling with black box constraints and DAGs. *Computer-Aided Design*, 75 :1–12, 2016.

[GHY04] Xiao-Shan Gao, Christoph M. Hoffmann, and Wei-Qiang Yang. Solving spatial basic geometric constraint configurations with locus intersection. *Computer-Aided Design*, 36(2) :111 – 122, 2004. Solid Modeling and Applications.

[GMMP11] F. Giannini, E. Montani, M. Monti, and J-P. Pernot. Semantic evaluation and deformation of curves based on aesthetic criteria. *Computer-Aided Design and Applications*, 8(3) :449–464, 2011.

[Hen92] Bruce Hendrickson. Conditions for unique graph realizations. *SIAM journal on computing*, 21(1) :65–84, 1992.

[HJA97] Christoph M Hoffmann and Robert Joan-Arinyo. Symbolic constraints in constructive geometric constraint solving. *Journal of Symbolic Computation*, 23(2-3) :287–299, 1997.

[HJA05] Christoph M Hoffmann and Robert Joan-Arinyo. A brief on constraint solving. *Computer-Aided Design and Applications*, 2(5) :655–663, 2005.

[HKP17] Hao Hu, Mathias Kleiner, and Jean-Philippe Pernot. Over-constraints detection and resolution in geometric equation systems. *Computer-Aided Design*, 90 :84 – 94, 2017. SI :SPM2017.

[IMS11] Remi Imbach, Pascal Mathis, and Pascal Schreck. Tracking method for reparametrized geometrical constraint systems. In *Symbolic and Numeric Algorithms for Scientific Computing (SYNASC), 2011 13th International Symposium on*, pages 31–38. IEEE, 2011.

[IMS15] Rémi Imbach, Pascal Mathis, and Pascal Schreck. A robust and efficient method for solving geometrical constraint problems by homotopy. *CoRR*, abs/1503.07901, 2015.

[ISM14] Rémi Imbach, Pascal Schreck, and Pascal Mathis. Leading a continuation method by geometry for solving geometric constraints. *Computer-Aided Design*, 46 :138–147, 2014.

[JASR99] Robert Joan-Arinyo and Antoni Soto-Riera. Combining constructive and equational geometric constraint-solving techniques. *ACM Transactions on Graphics (TOG)*, 18(1) :35–55, 1999.

[Jau01] Luc Jaulin. *Applied interval analysis : with examples in parameter and state estimation, robust control and robotics*, volume 1. Springer Science & Business Media, 2001.

[JTNM06] Christophe Jermann, Gilles Trombettoni, Bertrand Neveu, and Pascal Mathis. Decomposition of geometric constraint systems : a survey. *International Journal of Computational Geometry & Applications*, 16(05n06) :379–414, 2006.

[Kho12] Mostafa Khorramizadeh. An application of the Dulmage-Mendelsohn decomposition to sparse null space bases of full row rank matrices. In *International Mathematical Forum*, volume 7, pages 2549–2554, 2012.

[KMF14] Arnaud Kubicki, Dominique Michelucci, and Sebti Foufou. Witness computation for solving geometric constraint systems. In *Science and Information Conference (SAI), 2014*, pages 759–770. IEEE, 2014.

[Kon92] K. Kondo. Algebraic method for manipulation of dimensional relationships in geometric models. *Computer-Aided Design*, 24(3) :141–147, 1992.

[LGP+16] Z. Li, F. Giannini, J-P. Pernot, P. Véron, and B. Falcidieno. Re-using heterogeneous data for the conceptual design of shapes in virtual environments. *Virtual Reality*, pages 1–18, 2016.

[LP09] László Lovász and Michael D Plummer. *Matching theory*, volume 367. American Mathematical Soc., 2009.

[MF09] Dominique Michelucci and Sebti Foufou. Interrogating witnesses for geometric constraint solving. In *2009 SIAM/ACM Joint Conference on Geometric and Physical Modeling*, pages 343–348. ACM, 2009.

[MMS14] Mireille Moinet, Guillaume Mandil, and Philippe Serre. Defining tools to address over-constrained geometric problems in computer aided design. *Computer-Aided Design*, 48 :42–52, 2014.

[Owe91] John C Owen. Algebraic solution for geometry from dimensional constraints. In *Proceedings of the first ACM symposium on Solid modeling foundations and CAD/CAM applications*, pages 397–407. ACM, 1991.

[PFGL08a] J-P. Pernot, B. Falcidieno, F. Giannini, and J-C. Léon. A hybrid models deformation tool for free-form shapes manipulation. In *34th Design Automation Conference (ASME DETC08-DAC49524)*, pages 647–657, New-York, USA, 2008. ASME.

[PFGL08b] Jean-Philippe Pernot, Bianca Falcidieno, Franca Giannini, and Jean-Claude Léon. Incorporating free-form features in aesthetic and engineering product design : State-of-the-art report. *Computers in Industry*, 59(6) :626–637, 2008.

[Rao16] R Rao. Jaya : A simple and new optimization algorithm for solving constrained and unconstrained optimization problems. *International Journal of Industrial Engineering Computations*, 7(1) :19–34, 2016.

[RSV89] Dieter Roller, François Schonek, and Anne Verroust. Dimension-driven geometry in cad : a survey. In *Theory and practice of geometric modeling*, pages 509–523. Springer, 1989.

[Saa03] Yousef Saad. *Iterative methods for sparse linear systems*. SIAM, 2003.

[Ser91] David Serrano. Automatic dimensioning in design for manufacturing. In *Proceedings of the First ACM Symposium on Solid Modeling Foundations and CAD/CAM Applications*, SMA '91, pages 379–386, New York, NY, USA, 1991. ACM.

[SGM09] Christopher Solomon, Stuart Gibson, and Matthew Maylin. A new computational methodology for the construction of forensic, facial composites. *Computational forensics*, pages 67–77, 2009.

[SM06] Pascal Schreck and Pascal Mathis. Geometrical constraint system decomposition : a multigroup approach. *International Journal of Computational Geometry & Applications*, 16(05n06) :431–442, 2006.

[SS06] Pascal Schreck and Étienne Schramm. Using invariance under the similarity group to solve geometric constraint systems. *Computer-Aided Design*, 38(5) :475–484, 2006.

[SWI05] Andrew J Sommese and Charles W Wampler II. *The Numerical solution of systems of polynomials arising in engineering and science*. World Scientific, 2005.

[TSM+11] Simon EB Thierry, Pascal Schreck, Dominique Michelucci, Christoph Fünfzig, and Jean-David Génevaux. Extensions of the witness method to characterize under-, over-and well-constrained geometric constraint systems. *Computer-Aided Design*, 43(10) :1234–1249, 2011.

[VSR92] Anne Verroust, François Schonek, and Dieter Roller. Rule-oriented method for parameterized computer-aided design. *Computer-Aided Design*, 24(10) :531–540, 1992.